Storypoint Problem Exploration - datamanagement

September 3, 2024

1 Storypoint Prediction: Problem Exploration

1.1 Problem Statement

In modern agile development settings, software is developed through repeated cycles (iterative) and in smaller parts at a time (incremental), allowing for adaptation to changing requirements at any point during a project's life. A project has a number of iterations (e.g. sprints in Scrum). Each iteration requires the completion of a number of user stories, which are a common way for agile teams to express user requirements.

There is thus a need to focus on estimating the effort of completing a single user story at a time rather than the entire project. In fact, it has now become a common practice for agile teams to go through each user story and estimate its "size". Story points are commonly used as a unit of measure for specifying the overall size of a user story.

1.2 Problem Formulation

Input: A string of length N that contains a story's name and description $C = \{c_1, c_2, c_3, ..., c_n\}$. For each story, a set of text embeddings that contains features $E = \{e_1, e_2, e_3, ..., e_m\}$ extracted from C has been provided.

Output: A natural number P associated with the story point of that user story

1.3 Dataset Information

Text Embeddings: Text embeddings are a way to convert words or phrases from text into a list of numbers, where each number captures a part of the text's meaning. The dataset has been preprocessed and converted into two kinds of text embeddings. You can choose to work with either of them or both: - Doc2Vec: Input strings are transformed into fixed-length vectors of size 128. These vectors capture the semantic meaning of words and their relationships within a document. - Look-upTable: Input strings are transformed into fixed-length vectors of size 2264. These vectors are obtained via transforming each word in the input strings into an identifier number, then padded to the length of the longest sample.

Dataset Structure & Format: Storypoint Estimation Dataset is stored in 3 folders labeled *raw data*, *look-up*, and *doc2vec*. Within each folder are 3 CSV files for training, testing, validation. Each csv file has the following columns: - `issuekey` : The unique identifier for a story. - `storypoint`: The correct number of storypoint. - An embedding column (`embedding` or `doc2vec`) contains text embedding vectors. The raw data csv will not have this and instead contain two columns with story `name` and `description`.

1.4 Exploration

1.4.1 Raw data exploration

[ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.feature_extraction.text import CountVectorizer
```

Output exploration

[ ]:
```
# Import raw data from the CSV file

project_name = 'datamanagement'

all_data = pd.concat([pd.read_csv('data/' + project_name + '/' + project_name +␣
 ↪'_train.csv'),
                      pd.read_csv('data/' + project_name + '/' + project_name +␣
 ↪'_valid.csv'),
                      pd.read_csv('data/' + project_name + '/' + project_name +␣
 ↪'_test.csv')])

print('Check the shape of the dataset', all_data.shape)
```

```
Check the shape of the dataset (4030, 4)
```

[ ]:
```
all_data.drop(['issuekey'], axis=1, inplace=True)
all_data.head()
```

[ ]:
```
                                         title  \
0            transition git repositories stash
1                     finalize mission statement
2              open lsst software mailing lists
3               transition confluence questions
4  add derivativesbased optimizer measmultifit

                                         description  storypoint
0  transition gitolitemanaged repositories atlass…          10
1  proposed mission statement lsst software stack…           2
2  benefit making lsst software development open …           1
3  open confluence questions site interaction com…          10
4  see story points estimate remaining work code …          10
```

First, let take a look at the distribution of the story point:

Interpretation of Skewness Values:

- **Skewness** > 0: Right-skewed distribution.
- **Skewness** < 0: Left-skewed distribution.

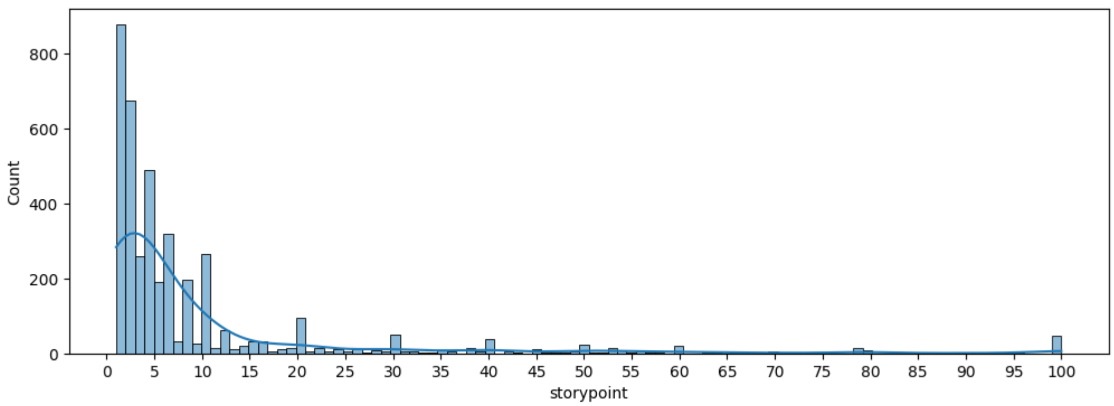
- **Skewness = 0**: Symmetrical distribution (like a normal distribution).

Interpretaion of kurtosis: - **Leptokurtic (Kurtosis > 3)**: The distribution has heavier tails and a sharper peak than the normal distribution. Data points are more likely to produce extreme values. The distribution has a higher peak and fatter tails. - **Platykurtic (Kurtosis < 3)**: The distribution has lighter tails and a flatter peak than the normal distribution. Data are fewer extreme values compared to a normal distribution. - **Mesokurtic (Kurtosis 3)**: The distribution has a similar kurtosis to the normal distribution, indicating a moderate level of outliers.

[ ]:
```
# Draw a histogram of the story points
plt.figure(figsize=(12, 4))
plt.xticks(np.arange(0, max(all_data['storypoint']) + 1, 5))
sns.histplot(all_data['storypoint'], bins=100, kde=True)

print('Skewness:', all_data['storypoint'].skew())
print('Kurtosis:', all_data['storypoint'].kurt())
```

```
Skewness: 3.5238904294540756
Kurtosis: 13.630513806599794
```



[ ]:
```
tmp = pd.concat([all_data['storypoint'].value_counts(),
                 all_data['storypoint'].value_counts() / all_data.shape[0] *␣
 ↪100],
                axis=1, keys=['Counts', 'Percentage (%)'])
tmp.head(20)
```

[ ]:

storypoint	Counts	Percentage (%)
1	876	21.736973
2	673	16.699752
4	490	12.158809
6	320	7.940447
10	266	6.600496
3	260	6.451613

8	196	4.863524
5	189	4.689826
20	95	2.357320
12	62	1.538462
30	51	1.265509
100	46	1.141439
40	38	0.942928
15	33	0.818859
16	32	0.794045
7	31	0.769231
9	27	0.669975
50	22	0.545906
60	20	0.496278
14	20	0.496278

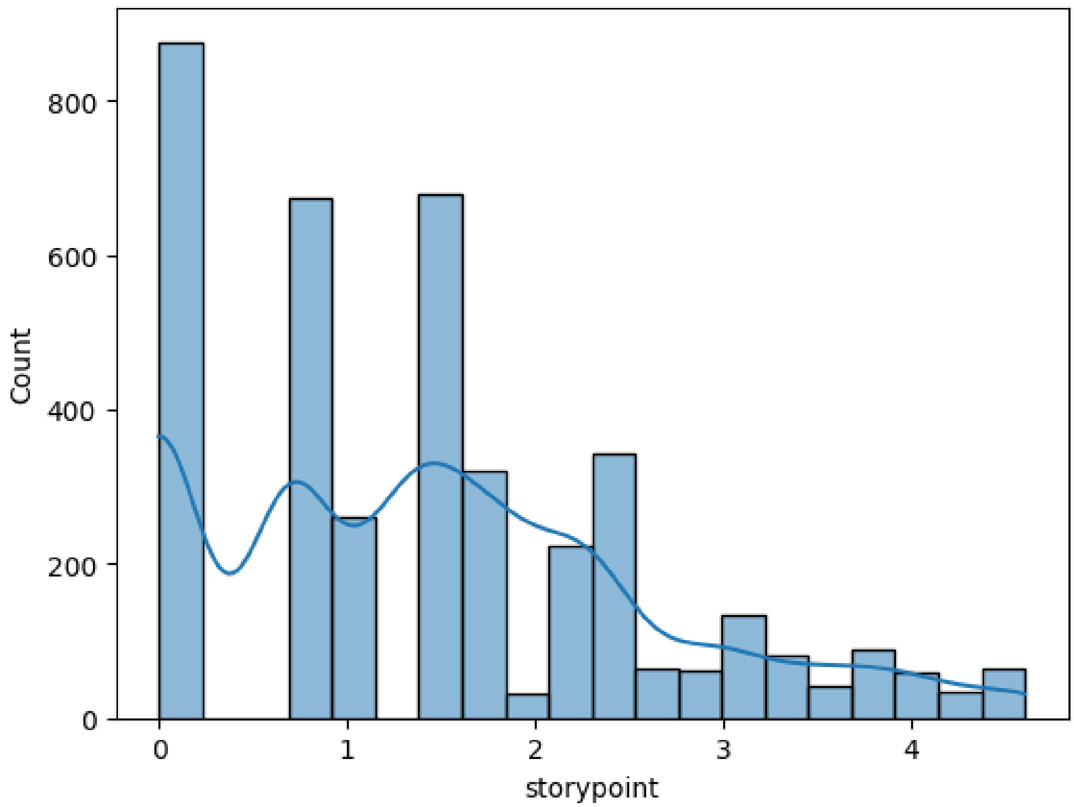
At the first sight, this data is bad. Then take a look at the statistic values, this data is even worse. Its distribution of the label is **right-skewed** and **leptokurtis**. This means if we use this to train model, the right side of the data can be the outliers and make the models become unsuable.

I will try 2 solutions: - Use log-scale on the label - Remove all the examples with label greater than a threshold (20, 30 or 40)

The first solution: logarithm magic

[ ]:
```
sns.histplot(np.log(all_data['storypoint']), bins=20, kde=True)
```

[ ]:
```
<Axes: xlabel='storypoint', ylabel='Count'>
```

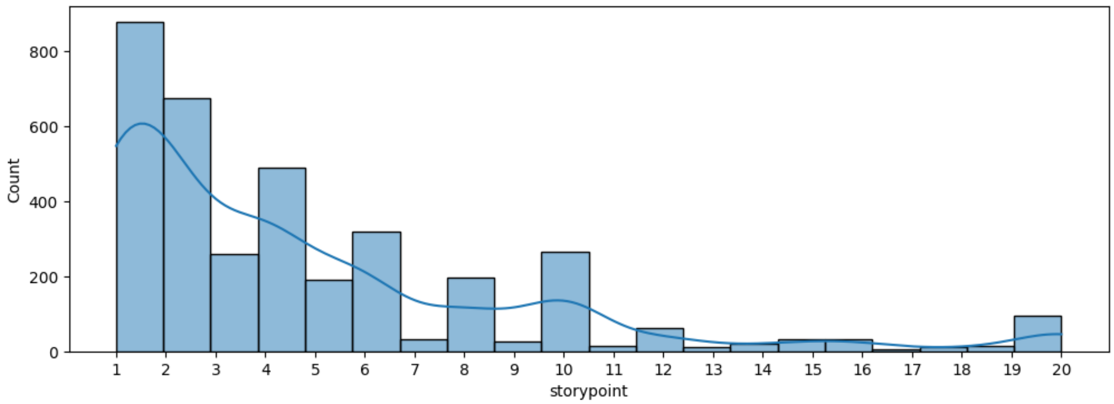


The second solution: Dismantle and Cleave

```
[ ]: threshold = 20 # This threshold means that we will take all the examples with␣
     ↪story points less than or equal to 20

     new_data = all_data[all_data['storypoint'] <= threshold]
     plt.figure(figsize=(12, 4))
     plt.xticks(np.arange(0, max(new_data['storypoint']) + 1, 1))
     sns.histplot(new_data['storypoint'], bins=threshold, kde=True)
     print('Fitered percentage: ', round(1 - new_data.shape[0] / all_data.shape[0],␣
     ↪2) * 100, '%')
```

```
Fitered percentage:  10.0 %
```



Input exploration The input of this problem is 2 texts: title and description. First we will find some statistics:

```
[ ]: title_lengths = all_data['title'].apply(lambda x: len(x.split(' ')))
print('Title analysis:')
print('   - Mean length:', round(title_lengths.mean()))
print('   - Min length:', title_lengths.min())
print('   - Max length:', title_lengths.max())

description_lengths = all_data['description'].apply(lambda x: len(x.split(' ')) ␣
 ↪if type(x) != float else 0)
print('Description analysis:')
print('   - Mean length:', round(description_lengths.mean()))
print('   - Min length:', description_lengths.min())
print('   - Max length:', description_lengths.max())
```

```
Title analysis:
   - Mean length: 5
   - Min length: 1
   - Max length: 15
Description analysis:
   - Mean length: 29
   - Min length: 0
   - Max length: 678
```

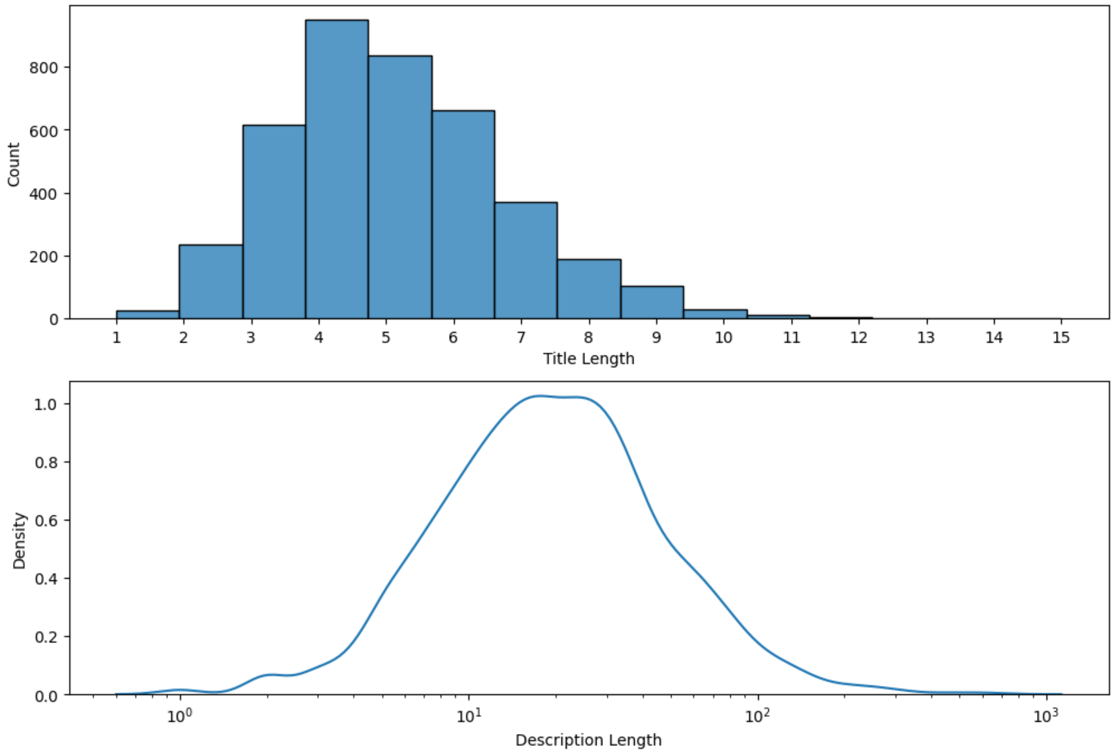
Plot the histogram of the title length and KDE of the description length (exclude 0):

```
[ ]: plt.figure(figsize=(12, 8))

plt.subplot(2, 1, 1)
plt.xticks(np.arange(0, max(title_lengths) + 1, 1))
plt.xlabel('Title Length')
sns.histplot(title_lengths, bins=max(title_lengths))
```

```
plt.subplot(2, 1, 2)
plt.xlabel('Description Length')
plt.xscale('log')
sns.kdeplot(description_lengths[description_lengths > 0])
```

[ ]: <Axes: xlabel='Description Length', ylabel='Density'>



I think we should check the correlation between title length and description length:

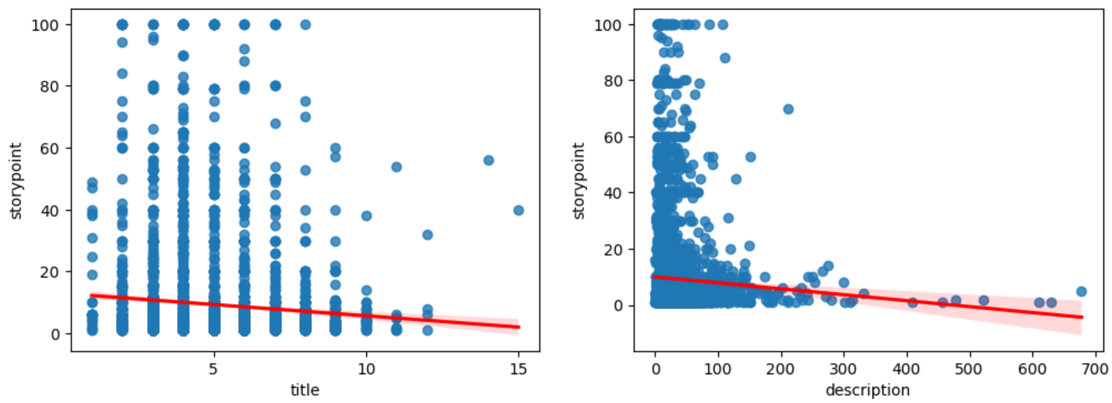
[ ]:
```
plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)
plt.xticks(np.arange(0, max(all_data['title'].apply(lambda x: len(x.split('␣
↪')))) + 1, 5))
sns.regplot(x=all_data['title'].apply(lambda x : len(x.split(' '))),
            y=all_data['storypoint'],
            line_kws={'color': 'red'})

plt.subplot(1, 2, 2)
sns.regplot(x=all_data['description'].apply(lambda x : len(x.split(' ')) if␣
↪type(x) != float else 0),
            y=all_data['storypoint'],
```

```
                line_kws={'color': 'red'})
```

[ ]: <Axes: xlabel='description', ylabel='storypoint'>



We can see correlation between title, description with storypoint (slightly slop down).

Let dive deeper in the input:

Title analysis:

```
[ ]: count_vectorizer = CountVectorizer()
count_vectorizer.fit(all_data['title'])

dictionary = pd.DataFrame(list(count_vectorizer.vocabulary_.items()),␣
 ↪columns=['word', 'frequency'])
dictionary.sort_values(by='frequency', ascending=False, inplace=True)
print(dictionary.shape)
dictionary.head(10)
```

(4292, 2)

[ ]:

	word	frequency
2537	zscalejava	4291
3669	zscale	4290
3178	zoom	4289
225	zookeeper	4288
1940	zone	4287
3613	zogy	4286
3546	zindex	4285
4079	zeropoint	4284
3329	zenodos	4283
3327	zenodiometadata	4282

Description analysis:

[ ]:
```
count_vectorizer = CountVectorizer()
count_vectorizer.fit(all_data[all_data['description'].isnull() ==␣
 ↪False]['description'])

dictionary = pd.DataFrame(list(count_vectorizer.vocabulary_.items()),␣
 ↪columns=['word', 'frequency'])
dictionary.sort_values(by='frequency', ascending=False, inplace=True)
print(dictionary.shape)
dictionary.head(20)
```

```
(14858, 2)
```

[ ]:

	word	frequency
8760	zscale	14857
7505	zpmp	14856
2533	zopeinterface	14855
11059	zooming	14854
10909	zoomed	14853
11051	zoom	14852
4784	zookeper	14851
2581	zookeeperlog	14850
674	zookeeper	14849
3350	zooerrorhandlesocketerrormsg	14848
7025	zonegrid	14847
6778	zone	14846
12777	zogy	14845
2162	znodes	14844
9071	zlibelx	14843
6005	zlib	14842
13108	zivezic	14841
12542	zindex	14840
4221	zhang	14839
4547	zfs	14838

Yet I don't find any thing special about the words in input except so many things are bad.

1.4.2 Solving strategies

My first intuitation in this problem is that the hard part is not on the algorithm we use, it is on the **embedding** part. Therefore, in case the given embedded datasets work not properly, I will use a better embedding method which is **Bidirectional Encoder Representations from Transformers (BERT)**. Also, I will try an old way to embedding the text too: **Bag of words**.

In conclusion, I will have 4 ways to embed the text: - doc2vec (already available) - Look up (already available) - Bag Of Words - BERT

About algorithm, I will try all the regression algorithm that may give a good result:

- Ridge Regressor
- Support Vector Regressor

- Random Forest Regressor
- Gradient Boosting
- XGBoost
- Lightgbm
- Blended

Maybe, we can change the problem to the classification problem with 100 labels (desparation confirmed). In the classification problem, I will use: - Support Vector Classifier - Softmax Regression (Multinomial Logistic Regression) - Random Forest - Adaboost - XGBoost

Thanks to the libaries, the implementation of all the algorithm shrinks to its minimum form.

At last, there is still a situation that all of mentioned model don't give a good result. This gamble is thrilling (hopeless).

"But would you lose?"

Nah, I'd win.